5780_Postlab_01

Shem Snow

January 2024

1 :

- **MODER:** The port mode register uses two bits to specify which of four different 'modes' the pins will use to send and receive data. 00 and 01 are digital input and output modes. 11 is analog mode (don't need to specify input or output because a real analog value exists on the pins). 10 is 'Alternate function' mode which uses election of internal peripherals on the board to determine its behavior.

  Without configuring this register, using GPIO pins as anything aside from digital inputs is not possible.

- **OTYPER:** The output type register selects the output mode for each pin. For pins without output mode configured, the bits in this register have no effect. There are two possible modes which we set with a single bit. 0 sets it to the "push-pull" mode which makes so a logical "1" pushes the pin to VCC and a "0" pulls it to ground. 1 sets it to "open-drain" mode which makes so a logical "0" pulls the pin to ground but a "1" leaves it floating.

- **OSPEEDR:** The output speed register in an ARM processor allows it to disable and/or reduces the speed of the peripherals (digital signal toggling speed). This helps it to save power. There are three different speed modes set by two bits. "x0", 01, and 11 are low, medium, and high respectively.

- **PUPDR:** The pull-up, pull-down register connects internal pull-up or pull-down "resistors" to a pin in order to prevent 'floating' voltages on them. These resistors are actually transistors that are not completely turned on but will leak enough current to act as a high-value resistor. There are two bits indicating if it is pull up or down. It would be a waste of power to have both and so the valuation "11" is reserved.

- **IDR:** The input data register has read-only bits. This makes so we can use it as a sensor to the outside world because the real-world signals trigger the logical state of the pins.

- **ODR:** The output data register can be digitally set to 1 or 0 and the pin will hold that state. This is useful for controlling 'off-board' devices.
- **BSRR:** The bit set/reset register is write-only. It's advantage is that you can set individual bits quickly and reliably. This makes so you don't have to read the register's value and construct and operate on a mask. The lower half of the register sets bits in the output bits and the upper half resets them.
- **LCKR** The configuration lock register locks the other configuration registers for the associated pin; this can prevent a malfunctioning program from accidentally changing a pin into an undesired mode. Once activated, you may not edit the locked register without first processing a timed sequence of writes on a specific bit.
- **GPIOx_AFRL/GPIOx_AFRH** Since every pin on the board has 4 bits that configure it, the alternate function low/high registers are used to configure alternate functions for all 16 pins. $16 * 4 = 64$ so that's why there are two of them.
- **BRR** The bit reset register is similar to the BSRR except instead of the clearing bits all being in the upper half of the register, they are in the lower half.

2 :

In the MODER register, write the value of 11 to the bits controlling a pin in order to set its port mode register to 'Analog mode'.

3 :

Two facts come into play here: First is that we start counting from zero. Second is that the upper half of the bits are used to reset bits in the register. This means the 20th bit is the one to set if I want to clear the first bit in the ODR.

4 :

- $0xAD + 0xC7 = 1010\_1101 + 1100\_0111 = 1110\_1111 = (239)_{10}$
- $0xAD \cdot 0xC7 = 1010\_1101 \cdot 1100\_0111 = 1000\_0101 = (133)_{10}$
- $0xAD \cdot \neg(0xC7) = 1010\_1101 \cdot \neg(1100\_0111) = 0010\_1000 = (40)_{10}$
- $0xAD \oplus 0xC7 = 1010\_1101 \oplus 1100\_0111 = 0110\_1010 = (106)_{10}$

5 :

To CLEAR a subset of bits within a register, you would use the bit-wise AND with a 'mask' that contains logical '1' in all the bits you do not wish to change/clear and contains 0 in the bits you do want to clear.

For the example provided (clearing the fifth and sixth bit), I would:

1. Depending on how close the two bits are, I would either do it in one or two parts. Since 5 and 6 are next to each other, I would just do them together in one part. So **set** $mask = 3$ (11_2 in binary).
2. Shift the bits to be in the position you want to alter. In the example **shift the mask left 5 times** i.e. $mask = mask \ll 5$ i.e. $mask = mask \cdot 2 \cdot 5$.
3. At this point, the mask holds a '1' in the positions we want to clear and a '0' everywhere else. I want the opposite so I would invert the mask i.e. $mask = \neg mask$.
4. Finally, clear the bits by ANDing the register with the mask. The '0's in the mask are said to 'gate' the AND gate which means they force it to a specific value (0 in this case).

6 :

As mentioned before, the output speed register (OSPEEDR) has three speeds (low, medium, and high). The maximum speed the STM32F072R8 GPIO pins can handle in the lowest speed setting is 1 MHz (from page 82 in the M0 manual).

7 :

- **TIM1 (TIMER1)** is enabled by RCC_APB2ENR_TIM1EN (line 7896 in stm32f072xb.h) which is part of the **RCC_APB2ENR register**.
- **DMA1** is listed in the AHB peripherals (lines 682-690). Line 7851 shows that the RCC_AHBENR_DMAEN enables the clock. That's part of the **RCC_APB2ENR register**.
- **I2C1** is enabled by RCC_APB1ENR_I2C1EN (line 7956) which is part of the **RCC_APB1ENR register**.